View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Missouri University of Science and Technology (Missouri S&T): Scholars' Mine

Missouri University of Science and Technology

Scholars' Mine

Computer Science Technical Reports

Computer Science

23 Jul 1993

Parallel Genetic Algorithm for the DAG Vertex Splitting Problem

Matthias Mayer

Follow this and additional works at: https://scholarsmine.mst.edu/comsci_techreports

 Part of the Computer Sciences Commons

Recommended Citation

Mayer, Matthias, "Parallel Genetic Algorithm for the DAG Vertex Splitting Problem" (1993). *Computer Science Technical Reports*. 43.
https://scholarsmine.mst.edu/comsci_techreports/43

This Technical Report is brought to you for free and open access by Scholars' Mine. It has been accepted for inclusion in Computer Science Technical Reports by an authorized administrator of Scholars' Mine. This work is protected by U. S. Copyright Law. Unauthorized use including reproduction for redistribution requires the permission of the copyright holder. For more information, please contact scholarsmine@mst.edu.

Parallel Genetic Algorithm for the DAG Vertex Splitting Problem

Matthias Mayer[1]

CSC-93-21

Fri Jul 23 1993

Department of Computer Science
University of Missouri-Rolla
Rolla, MO 65401, U.S.A.
(314) 341-4491

[1]electronic mail address: matze@cs.umr.edu

Parallel Genetic Algorithm for the DAG Vertex Splitting Problem

Matthias Mayer *
Department of Computer Science
University of Missouri-Rolla
Rolla, MO 65401
USA

July 23, 1993

Abstract

Directed Acyclic Graphs (DGAs) are often used to model circuits and networks. The path length in such DAGs represents circuit or network delays. In the vertex splitting problem, the objective is to determine a minimum number of vertices from the graph to split such that the resulting graph has no path of length greater than a given maximum delay δ. The problem has been proven to be NP-hard. A sequential Genetic Algorithm has been developed to solve the DAG Vertex Splitting Problem. Unlike a standard Genetic Algorithm, this approach uses a variable chromosome length to represent the vertices that split the graph and a dynamic population size. A parallel version of the sequential Genetic Algorithm has been developed. It uses a fully distributed scheme to assign different string lengths to processors. A ring exchange method is used in order to exchange "good" individuals between processors. Almost linear speed-up and two cases of super linear speed-up are reported.

Keywords: Parallel Genetic Algorithm, variable string length, dynamic population size, vertex splitting

*email: matze@cs.umr.edu

1 Introduction

Genetic Algorithms (GAs)[Holland75] are adaptive search techniques that have been shown to be robust optimization algorithms. In contrast to other optimization techniques, Genetic Algorithms base their progress on the performance of a population of candidate solutions, rather than on a single candidate solution. GAs are loosely based upon Darwin's principle of natural selection and natural genetics. They have become increasingly popular in recent years as a method for solving combinatorial optimization problems [Goldberg89a].

Many applications [Goldberg89a] involving computer networks and electrical circuits can be modeled as a graph, and the path lengths in these graphs represent circuit or network delays. If the path lengths in a Directed Acyclic Graph are too long and need to be reduced, the reduction can be done by splitting certain vertices in the graph into two vertices which results in a reduction of the path length. The DAG Vertex Splitting Problem [Paik90] is an optimization problem in which the smallest number of vertices in the graph have to be found such that the longest path in the split graph is less than or equal a pre-specified maximum called δ. The DAG Vertex Splitting Problem addressed in this paper has many applications in the fields of computer science and electrical engineering, such as finding the minimum number of placements of signal boosters in a network, where the computers in the network are represented by the vertices of the graph, or the placement of flip-flops in partial scan designs. Heuristics [Paik90] have been used earlier to solve the DAG Vertex Splitting Problem. The DAG Vertex Splitting Problem has been identified as NP-hard [Paik90].

2 The DAG Vertex Splitting Problem

The DAG vertex splitting problem (DVSP) can be stated as follows [Paik90]: Let G = (V, E, w) be a Weighted Directed Acyclic Graph (WDAG) with vertex set V, edge set E, and edge function w. w(i, j) is the weight of the edge $< i, j > \in E$. w(i, j) is a positive real number for $< i, j > \in E$ and is undefined if $< i, j > \notin E$. If w(i, j) = 1 $\forall < i, j > \in E$ then the graph has unit weights and is called a Directed Acyclic Graph (DAG). It has been proven in Paik et.al. [Paik90] that finding a solution for the DVSP is NP-hard for graphs with unit weights and $\delta \geq 2$. Since Directed Acyclic Graphs are just a special case of Weighted Directed Acyclic Graphs these results also apply to the WDAG [Paik90]. The delay, $d(P)$, on the path P, is the sum of the weights of all the edges on that path P. The delay, $d(G)$, of the graph G is the maximum path delay in the graph.

Let G/X denote the WDAG that results when each vertex v in X is split into two vertices v^i and v^o such that all outgoing edges $< v, j > \in E$ are replaced by edges of the form $< v^o, j >$ and all incoming edges $< i, v > \in E$ are replaced by edges of the form $< i, v^i >$. Outbound edges of v now leave v^o, while inbound edges of v now enter v^i. Figure 1 shows an example of a DAG without/with vertex 3 split.

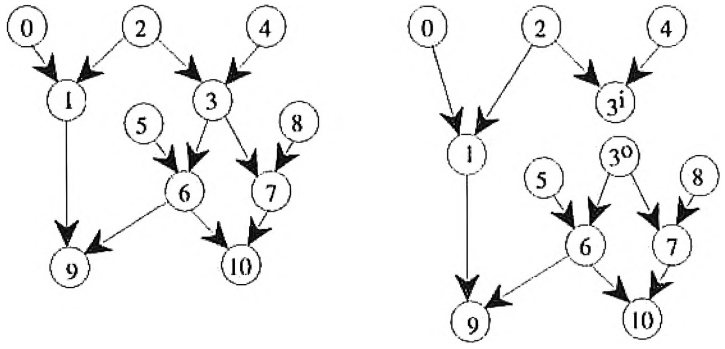

Figure 1: Example of DAG without/with vertex 3 split

The DAG Vertex Splitting Problem is to find the least cardinality vertex set $X \ni d(G/X) \leq \delta$, where δ is a pre-specified maximum delay. For the DAG of Figure 1 and $\delta = 2$, X = {3} is the best solution to the DVSP.

3 The Genetic Algorithm for the DVSP

The objective for the GA is to find a minimal set of vertices that split the graph such that the resulting graph has no path of length $> \delta$. The strings (chromosomes) in each individual of the population represent the set of splitting vertices (or split set for short) that are used to split the graph [Mayer93a, Mayer93b]. Thus, the smaller the split set the better the solution. Theoretically, every vertex in the graph, excluding source and sink vertices, can be split and a vertex can only occur once in the split set. But some of these vertices might not be on a path whose length is greater than δ. Thus, it would be useless to consider them as potential vertices to split. A new set is introduced, called *potential vertices*, which contains only those vertices that are on paths whose length is greater than δ.

The Genetic Algorithm starts the search with an initial string length and continues with multiple rounds of optimization. Each optimization round tries to find a feasible solution with a fixed size split set (string length), i.e. the vertices in the split set of at least one individual can split the graph in such a way that the resulting graph has a maximum path length which is less than or equal δ. If a solution is found in a particular round, the next round attempts to shorten the string length and find a new solution with fewer vertices. Therefore, the string length varies over time. Variable string length has been used before in GAs [Goldberg89b, Goldberg90]. The Genetic Algorithm works only on one certain string length at a time. Different string lengths within a given population are not allowed simultaneously because it makes the GA more complex when applying the select and crossover functions.

The algorithm works as follows:

1. try to find a suboptimal solution by splitting x vertices
2. if a suboptimal solution is found, reduce the number of vertices and try again
3. if no solution has been found within a certain number of generations, expand the number of vertices and try again.

The Genetic Algorithm starts the search for an optimal solution with an initial string length for the initial population. A solution to the DVSP exists, if all vertices in the potential vertex set are split [Mayer93b]. Thus, the initial string length for the GA would be the cardinality of the set of potential vertices. Since this number might be far away from the global optimum, a method called *Binary Approximation (BA)* was developed, to narrow down the area around the global optimum in order to quickly find a better initial string length. The function works as follows:

> The solution to the DVSP has to lie somewhere between splitting one vertex and splitting all potential vertices. Thus, a binary search is started midway between these two boundaries. A certain number of split sets (strings), determined by the parameter *number of tries for BA*, are created randomly with a string length halfway between the lower and the upper bound. If a solution is found among these split sets, it is marked as a new upper bound for the BA. If no solution can be found then this is assumed to be a lower bound. This process is repeated until the difference between the lower and the upper bound is less than one. After termination, the upper bound is assigned to the initial string length.

The *select* function is a standard select using the roulette wheel. Instead of a linear search through the population, a binary search has been implemented which returns

the index of the selected individual. Since the goal of optimization is to minimize the longest path in the graph by splitting vertices, the fitness function is defined to be $\frac{1}{Longest\ Path}$.

It is possible that the DVSP has a solution by splitting only one vertex. This means that the string length is only one. If the crossover function is performed on individuals with a string length of one, no new individuals are introduced into the population. Only mutation can introduce new individuals. Since the mutation rate is usually very low, there exists a high probability of missing a solution with one vertex. A function called *Take care of ones* is used to eliminate this possibility. This function tries every vertex in the potential vertex set one at a time to check if there exists a solution. This function is used before the GA is started. If this function finds a solution by spitting only one vertex, then there is no need to start the GA.

The *uniform crossover* function [Syswerda89, Spears91] is used to generate offspring from the parents. The uniform crossover was shown to outperform the one-point and two-point crossover in most cases [Spears91]. Every applied crossover results in two offspring. The offspring are placed into the population together with its parents which results in a temporary increase of the population. This population is reduced later by the *recombination* function. This way of creating a new generation was chosen to ensure that less fit offspring do not overwrite a more fit parent.

The *mutation* function operates only on the newly created offspring. Once a vertex has been chosen for mutation it is replaced by a new vertex picked from the potential vertex set.

The *recombination* function takes the old population and the new offspring and reduces it down to the previous population size. The reduction is based on the select function which ensures that fit individuals have a higher probability of survival into the new generation. This means that individuals of the old population can survive into

the new population while new offspring may die, depending on their fitness values. This method ensures that a highly fit parent does not get eliminated by a lower fit offspring. The recombination function also takes care of a certain variety in the new generation by making sure that no individual (parents and offspring) gets selected more than once for the new population.

4 The Parallel Genetic Algorithm

A Parallel Genetic Algorithm (PGA) for the DVSP [Mayer93b] has been developed for an Intel Hypercube iPSC/2 with a maximum of 16 processors such that it can run on any number of processors ≥ 2. The basic idea behind the PGA is to distribute the available processors over the search space and let each processor operate on a subpopulation of the total population. In other words, every processor operates on a different string length. The search space is virtually reduced by increasing the lower bound of the search space if no solutions have been found within a certain number of generations. After the search space has been reduced, the processors are distributed again. This continues until the upper and lower bound of the search space are equal. If the virtual search space becomes smaller than the number of available processors, some processors are assigned the same string length. If processors operate on the same string length they can exchange "good" individuals and replace "bad" individuals in their subpopulation by the "good" individuals they receive. A different exchange strategy than described in Tanese [Tanese87] is used. Tanese's exchange model uses pairs of neighbors to exchange individuals. This does not work in this model because there might be an odd number of processors working on one string length which does not allow to create pairs. Therefore, a *ring exchange* model is used. For example, assume processor 0, 6, and 12 are assigned to work on the same string length. When the processors exchange individuals, processor 0 sends its individuals to

processor 6, processor 6 sends its individuals to processor 12, and processor 12 sends its individuals to processor 0. "Good" individuals in each subpopulation are selected probabilistically using the *select* function. The "bad" individuals that are replaced by the "good" individuals are selected by using an *inverse select* that uses the inverse of the fitness value to give "bad" individuals a higher probability of getting selected.

5 Experimental Results

The graphs used in the experiments for testing the Genetic Algorithms were derived from the ISCAS-85 benchmark combinational circuits [Brglez85]. The vertices in the DAG model represent the gates in the circuit and the edges correspond to connections between gates. The delay (weight) for all the edges was set to one. Most of the experiments were run on one graph (C432) which has 196 vertices and 336 edges. The longest path in that graph is 17. It was tested with $\delta = 5$, which results in 153 potential vertices. The total search space is $2^n - 1$, where n is the number of potential vertices [Mayer93b]. Thus, the total search space for the graph C432 is $2^{153} - 1 = 1.142 \cdot 10^{46}$.

5.1 Binary Approximation

An experiment was conducted on the Binary Approximation to see how the parameter *number of tries for BA* influences runtime and the initial string length. The following parameter settings were used:
number of tries for BA = 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500.
Each run was repeated 20 times to get a good average. The tests were run on all available graphs. All experiments exhibited a similar behavior: a fast jump down to a small *initial string length*, when the *number of tries for BA* = 10 and only smaller reductions in the *initial string length* when the *number of tries for BA* is higher. The

run time for the Binary Approximation increases lineary with a linear increase in the *number of tries for BA*. Figure 2 shows the results for graph C432. The percentage of reduction can be computed according to the following formula:

$$reduction = 100 - (\frac{initial\ string\ length}{total\ number\ of\ potential\ vertices}) \cdot 100$$

For graph C432 the reduction lies between 50% and 60%. The other tested graphs showed a reduction of mostly between 80% and 90% [Mayer93b].

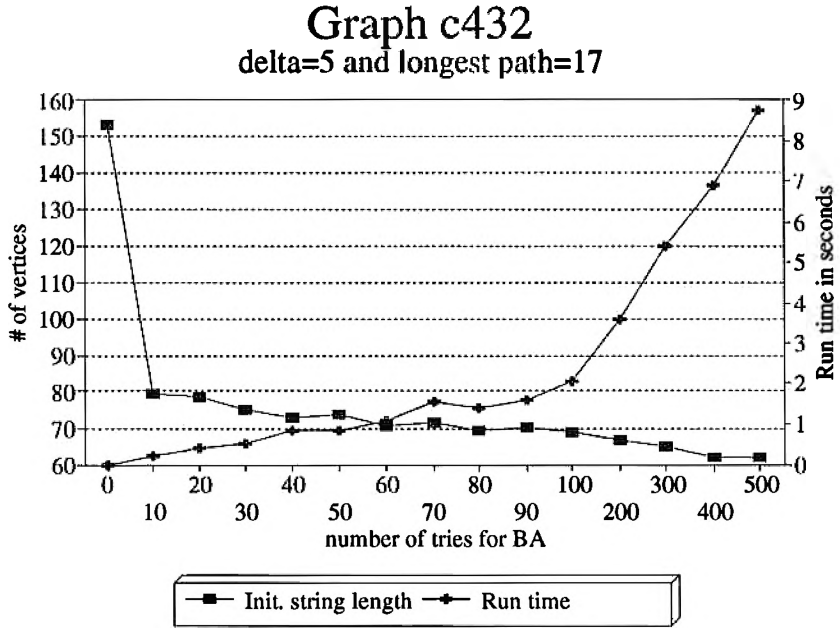

Figure 2: Initial string length reduction with Binary Approximation on graph C432

5.2 Speed-Up

The speed-up is one of the most important performance measures for the parallel algorithm. It is computed according to the following equation:

$$speed - up = \frac{Run\ time\ of\ best\ sequential\ algorithm}{Run\ time\ of\ the\ parallel\ algorithm\ using\ P\ processors}$$

Thus, the better the speed-up, the better the parallel algorithm. Linear speed-up occurs if speed-up = # processors. However, there exists a problem with the above

equation, since the best sequential Genetic Algorithm is not known. Therefore, the equation needs to be modified in order to calculate the speed-up. Usually the run time of the sequential algorithm is measured on one processor. Therefore, the speed-up is

$$speed - up = \frac{Run\ time\ of\ one\ processor}{Run\ time\ of\ the\ parallel\ algorithm\ using\ X\ processors}$$

Since the described PGA is designed to work on multiple processors, the run time comparison between the PGA with just one processor and the PGA with more than one processor is not possible. Therefore, linear speed-up is assumed between one processor and two processors and the speed-up is calculated using the formula below:

$$speed - up = \frac{Run\ time\ of\ two\ processor \cdot 2}{Run\ time\ of\ the\ parallel\ algorithm\ using\ X\ processors}$$

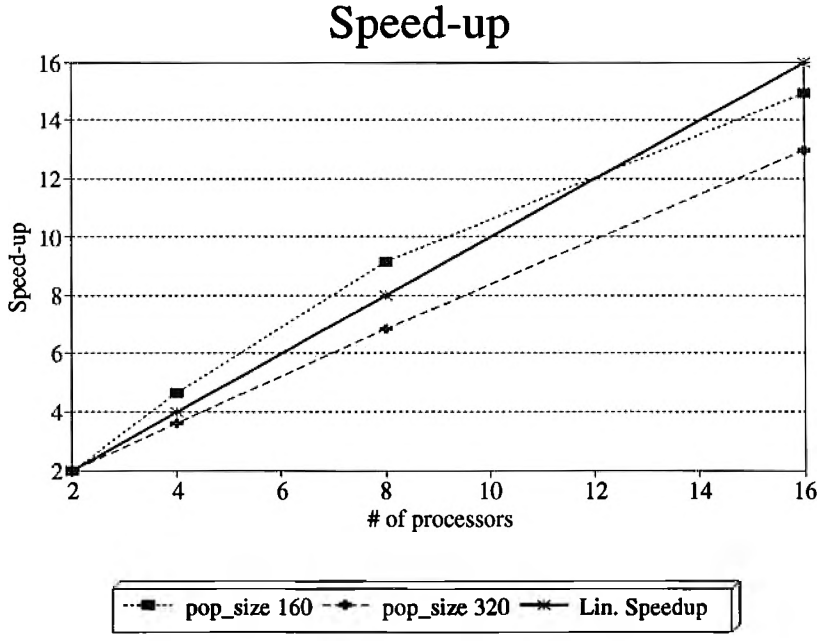

Figure 3: Speed-up of the Parallel Genetic Algorithm

The experiment to measure the speed-up was run on two different total population sizes, 160 and 320. Crossover and mutation rate were set to 0.6 and 0.005 respectively. These settings were chosen because they got the best results in previous experiments

[Mayer93a]. The number of individuals to be exchanged between the processors was constantly set to be 10 and each test was run 10 times to get an average. It can be seen from Figure 3 that the PGA with a total population size of 320 yields almost linear speedup.

The reason why the algorithm is below the linear speed-up line is because of the communication overhead for synchronizing the processors and exchanging individuals. The PGA with 160 individuals in the total population shows what is called super linear speed-up for four and eight processors. This is due to the fact that Genetic Algorithms are probabilistic algorithms and therefore they can sometimes find better solutions faster. Figure 4 and Figure 5 show the graphs for total population sizes of 160 and 320 respectively with respect to the number of processors. In most cases better solutions were found with a higher number of processors. The reason for this behavior can be explained as follows: Dividing the total population among more processors means that every processor has fewer individuals in its local subpopulation. With fewer individuals in the subpopulation it is easier to select superior individuals. On the other hand, if the subpopulation size gets too small, it is harder to create offspring with new genes because of the smaller amount of genetic material in the subpopulation. This result is even more important than achieving a linear speed-up for the Parallel Genetic Algorithm. Therefore, the use of the PGA has two big advantages:

1. smaller run time with more processors

2. better solutions with more processors

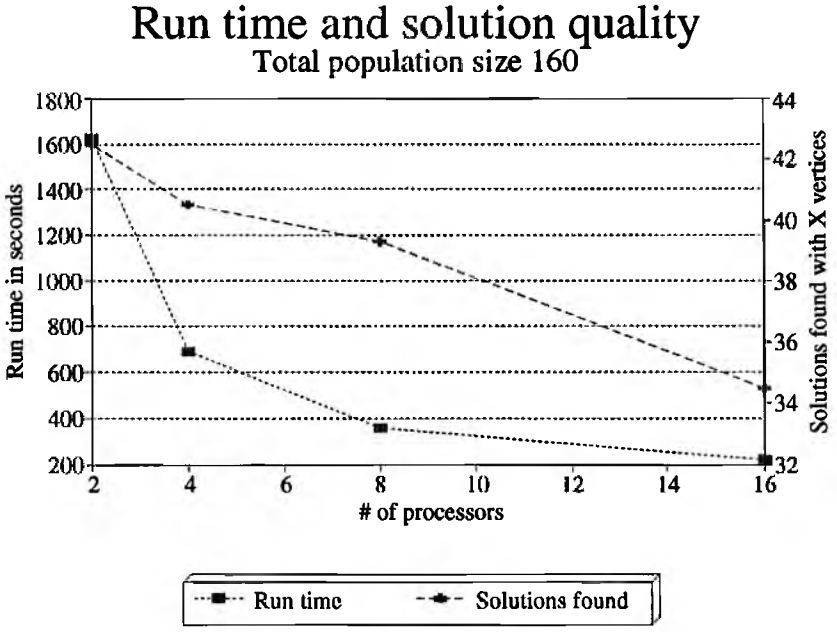

Figure 4: Run time and solution quality with different number of processors and a total population size of 160

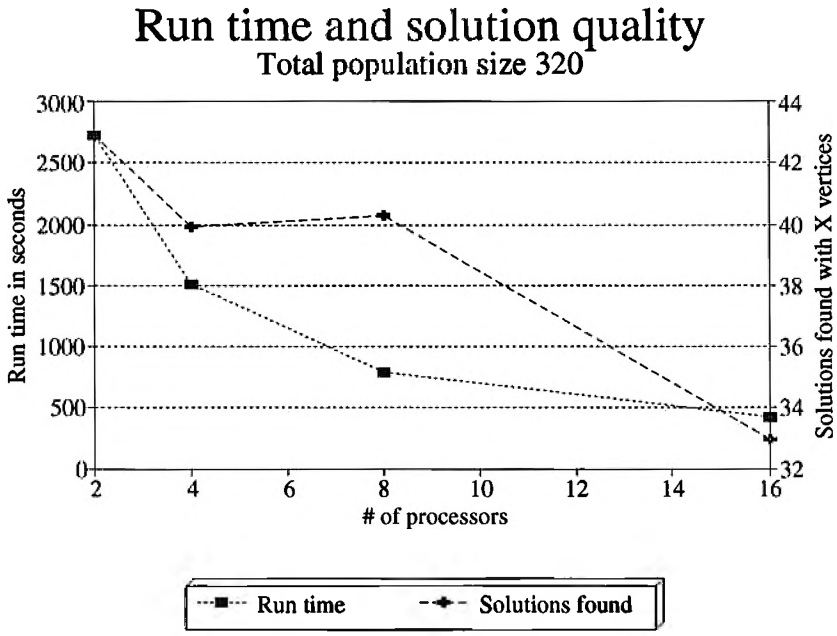

Figure 5: Run time and solution quality with different number of processors and a total population size of 320

6 Conclusions

Adaptive search algorithms are capable of adjusting efficiently to their environments. Nature has proven that it is a very good adaptive system. Therefore, search algorithms can be enhanced if natural behavior can be included into them. Genetic Algorithms are such adaptive search algorithms that are based on natural behavior.

This paper introduced the DAG Vertex Splitting Problem (DVSP) and described a Parallel Genetic Algorithm to solve it. A set of experiments was run to determine the behavior of the Binary Approximation. It is used to determine the *initial string length* for the initial population. It tries to find solutions by randomly picking vertices. The experiments indicate that even a small number of random tries can reduce the initial string length by up to 90%. Another set of experiments was run on the PGA to determine the speed-up of the PGA. A super linear speed-up is reported for four and eight processors for a total population size of 160. An almost linear speed-up is obtained for runs with a population size of 320.

Some heuristics [Paik90] have been proposed to solve the DVSP. Future research will incorporate some of these heuristics into various functions of the GA, like select, crossover and mutate.

References

[Brglez85] Brglez, F. and Fujiwara, H., "A Neutral Netlist of Ten Combinatoral Benchmark Circuits and a Target Translator in FORTRAN", Proceedings IEEE Symposium on Circuits & Systems, pp. 663-666, 1985.

[Goldberg89a] Goldberg, D.E., "Genetic Algorithms in Search, Optimization & Machine Learning". Addison-Wesley, 1989.

[Goldberg89b] Goldberg, D.E., Korb, B., Deb, K., "Messy Genetic Algorithms: Motivation, Analysis and First Results", Complex Systems 3, pp. 493-530, 1989.

[Goldberg90] Goldberg, D.E., Deb, K., Korb, B., "Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale", Complex Systems 4, pp. 415-444, 1990.

[Holland75] Holland, J.H., "Adaption in natural and artificial systems", Ann Arbor: The University of Michigan Press, 1975.

[Mayer93a] Mayer, M. and Ercal, F., "Genetic Algorithms For Vertex Splitting in DAGs", to appear in *Proceedings of the 5th International Conference on Genetic Algorithms.* Also, Technical Report TR93-02. University of Missouri-Rolla, 1993.

[Mayer93b] Mayer, Matthias, "Parallel Genetic Algorithms for the DAG Vertex Splitting Problem", Master's Thesis, University of Missouri-Rolla, 1993.

[Paik90] Paik, D., Reddy, S. and Sahni, S., "Vertex Splitting in Dags and Applications to Partial Scan Design and Lossy Circuits". Technical Report TR-90-034, University of Florida, 1990.

[Spears91] Spears, W.M. and Anand, V., "A Study of Crossover Operators in Genetic Programming", Sixth International Symposium on Methodologies for Intelligent Systems. Charlotte, NC, pp. 409-418, 1991.

[Syswerda89] Syswerda, G., "Uniform Crossover in Genetic Algorithms", Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufman Publishers, pp. 2-9, 1989.

[Tanese87] Tanese, R., "Parallel Genetic Algorithm for a Hypercube", Proceedings of the Second International Conference on Genetic Algorithms, pp. 177-183, 1987.